JCMS - Développement orienté module avec Eclipse et SVN

A propos de ce document

Historique

0.1	09/05/12	Xuan Tuong LE	Création et rédaction du document
0.2	21/05/12	Olivier DEDIEU	Relecture du document
0.3	08/06/12	Xuan Tuong LE	Mise à jour du document
0.4	24/08/12	Olivier DEDIEU	Relecture du document
0.5	19/09/12	Xuan Tuong LE	Mise à jour du document
1.0	13/12/12	Xuan Tuong LE	Mise à jour du document. Document officiel diffusable

Contacts

Jalios SA
58 rue Pottier
78150 Le Chesnay

Si vous avez des questions ou souhaitez des éclaircissements sur ce document, vous pouvez nous contacter :

Service commercial

Jean-François Pellier

Tél : 01 39 23 31 15
E-mail : jean-francois.pellier@jalios.com

Service technique

Xuan Tuong LE

Tél : 06 14 97 44 44
E-mail : xuan-tuong.le@jalios.com

Sommaire

A propos de ce document 2
Historique 2
Contacts 2
Service commercial 2
Service technique 2
Sommaire 3
1. Objectif du document 4
2. Principes 5
2.1. Architecture 5
2.2. Mode opératoire 6
2.3. Les acteurs et leurs rôles 7
3. Mise en place 8
3.1. Configuration initiale 8
3.1.1. Serveur de recette 8
3.1.2. Serveur SVN 8
3.1.3. Eclipse et SVN 10
3.1.4. Check-in initial 11
3.1.4.1. Check-in initial des modules 12
3.1.4.2. Check-in initial des configurations et des données 14
3.1.4.3. Check-in initial des wars JCMS et modules associées 15
3.1.5. Configuration initiale de l'environnement de développement 16
3.1.5.1. Checkout initial des projets modules spécifiques JCMS 16
3.1.5.2. Checkout initial des configurations et des données 17
3.1.5.3. Création d'un projet web dynamique JCMS 17
3.2. Développement 25
3.2.1. Synchronisation entre le projet module JCMS et la webapp JCMS 25
3.2.2. Jalios Plugin Tools 25
3.2.2.1. Configuration du projet module JCMS 25
3.2.2.2. Configuration du projet webapp JCMS 26
3.2.2.3. Tester la synchronisation 26
3.2.2.4. Limites 27
3.2.3. Gestion du fichier custom.prop 28
3.2.4. Synchronisation des données du store 28
3.3. Recette 33
3.4. Mise en production 33
3.5. Annexe 34
3.5.1. FAQ 34
3.5.2. Usage du Store Merge 34

1. Objectif du document

L'utilisation d'un système de gestion de versions tel que SVN permet de centraliser la gestion des sources pour tous les développeurs et de conserver une trace de toutes les modifications apportées lors des développements.

Cet article présente l'architecture recommandée et les processus à appliquer pour développer des modules JCMS avec un serveur SVN.

Pré requis :

- Connaissance du mécanisme de module dans JCMS
- Connaissance du processus de déploiement d'une webapp JCMS
- Notion de base sur la mise en œuvre et l'utilisation d'un serveur SVN
- Installation et utilisation d'Eclipse

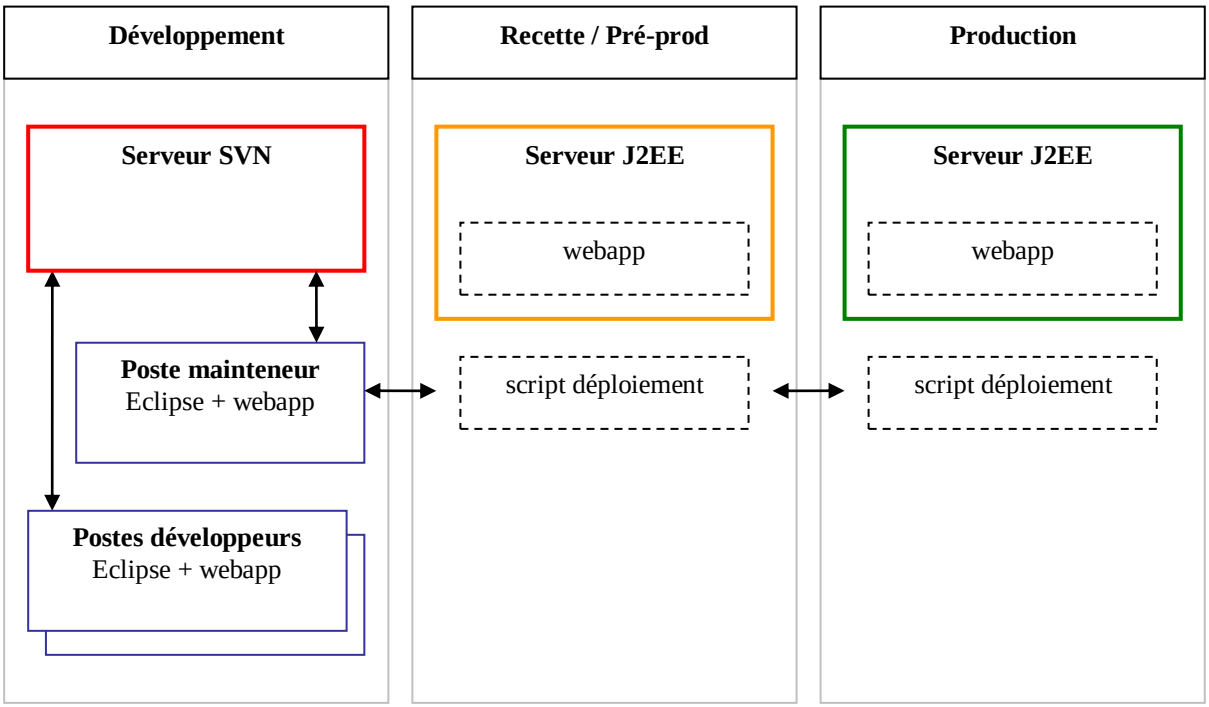
.

2. Principes

2.1. Architecture

L'ensemble des composants nécessaires sont :

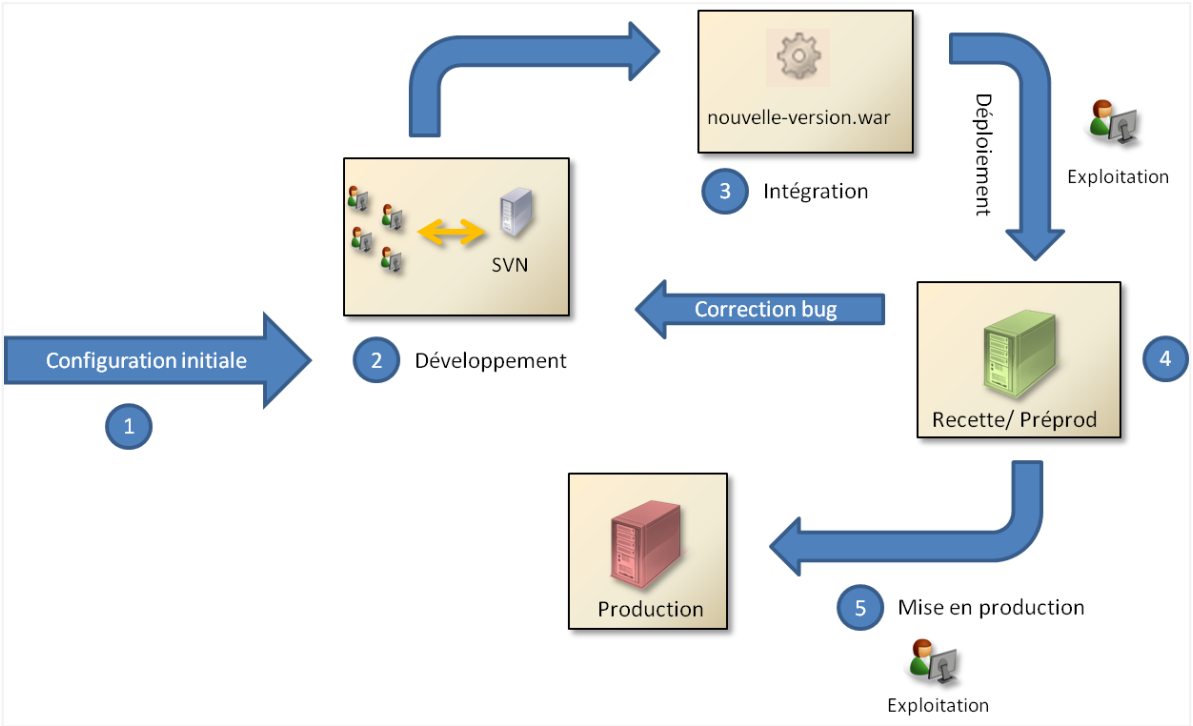
1. un serveur SVN pour les développements
2. un serveur J2EE (e.g. Tomcat) pour l'environnement de recette et de pré-production hébergeant la webapp JCMS. Un script de déploiement présent sur l'environnement de recette et celui d'exploitation pour les opérations de déploiement.
3. un serveur J2EE pour l'environnement de production hébergeant la webapp JCMS. Un script de déploiement est présent sur le système d'exploitation pour les opérations de déploiement.



2.2. Mode opératoire

Le processus de développement se décompose en 5 étapes :

1. **Configuration initiale**
   - **Serveur de recette** : Installation de la webapp JCMS et script de déploiement
   - **Serveur SVN** :
     - Création d'un repository vierge
     - Création d'une arborescence pour gérer les composants de développements (les modules, les configurations, etc...)
   - **Checkin initial** : Mettre à disposition des webapps cibles JCMS à développer sur SVN
2. **Développement**
   - Synchronisation avec le SVN, puis commit des développements.
3. **Intégration**
   - Construction de la livraison pour l'environnement cible (recette, préproduction, production) suivant les préconisations de Jalios (déploiement par .war)
4. **Recette**
   - Déploiement de la nouvelle webapp sur le serveur de recette et validation.
5. **Mise en production**
   - Déploiement de la même webapp validée par la recette sur le serveur de production.



2.3. Les acteurs et leurs rôles

On distingue quatre rôles principaux dans le processus de développement :

- **Le mainteneur / Responsable développement**
  C'est l'administrateur des développements, il est responsable de :
  - la construction de la webapp à livrer en se basant sur les développements dans SVN
  - la mise à jour de l'environnement de recette à partir du serveur SVN (déploiement).
- **Les développeurs**
  Ils récupèrent leur environnement de développement sur le serveur SVN (checkout ou synchronisation) et y committent leurs modifications.
- **L'équipe de recette**
  Ils vérifient que l'application est conforme aux attentes formulées.
- **L'équipe d'exploitation**
  Ils s'occupent de la mise en pré-production et production. Les exploitants supervisent l'état de production d'un parc d'applications de manière générique sans entrer dans le détail de fonctionnement de chaque application.

Qui fait quoi ?

	Mainteneur	Développeurs	Recette	Exploitation
1. Configuration initiale				
Serveur de recette	✓	-	-	-
Serveur SVN		-	-	✓
Checkin initial	✓	-	-	-
Checkout initial	-	✓	-	-
2. Développement	-	✓	-	-
3. Intégration	✓			
4. Recette	✓	✓	✓	
5. Mise en préproduction ou production		-	-	✓

3. Mise en place

Cette section détaille le mode opératoire pour configurer un environnement de travail d'équipe pour un projet JCMS.

3.1. Configuration initiale

3.1.1. Serveur de recette

Objectifs : Installation de la webapp JCMS en utilisant le script de déploiement

Installez sur le serveur de recette :

- La toute première version de la webapp
- Le script de déploiement : consultez l'article dédié sur notre site de support http://support.jalios.com/howto/deploy

Pour tous les détails d'installation de JCMS, consultez le manuel d'installation et d'exploitation de la version JCMS correspondante.

3.1.2. Serveur SVN

Objectif 1 : Création d'un repository SVN vierge. Cette tâche est généralement effectuée par l'exploitation gérant les serveurs.

Exemple, sur un serveur Linux.

Pré-requis (à effectuer par l'administrateur de la machine) :

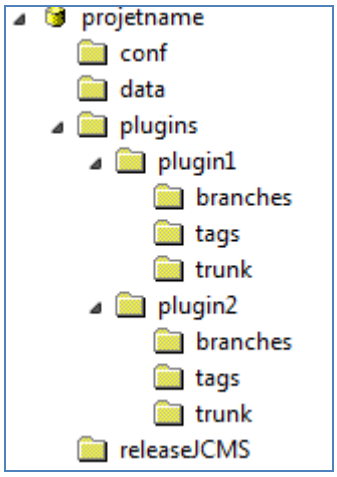
- Présence d'un compte utilisateur "svn" ainsi que d'un groupe du même nom. Tous les utilisateurs du SVN (mainteneur et développeurs) devront disposer de ce compte ou d'un compte utilisateur appartenant au groupe "svn".
- Création d'un dossier *"/home/svn/repositories"*. Ce dossier contiendra les repositories SVN.
- Lien symbolique de *"/svn"* vers *"/home/svn/repositories"*. Ceci permettra de simplifier l'accès aux différents repositories avec les outils de ligne de commande mais également depuis l'environnement Eclipse.

Création d'un nouveau repository "jcms" :

```
[svn@svn-server~]$ svnadmin create /home/svn/repositories/jcms
[svn@svn-server~]$ svn mkdir file:///svn/jcms/trunk file:///svn/jcms/branches
file:///svn/jcms/tags -m "creating initial repository layout"
```

Objectif 2 : Création d'une arborescence pour gérer les composants de développements. L'objectif est d'organiser pour faciliter le travail des développements et l'intégration :

- Sur un projet, un nouveau développeur doit configurer rapidement son environnement de développement à partir de SVN afin de démarrer ses développements.
- L'organisation du référentiel SVN doit répondre également aux objectifs d'intégrations. L'équipe d'intégration doit être capable de monter un processus de construction (ou "*build*") de livraison à partir de SVN. En pratique, le processus de construction est un script Ant. A noter qu'il est peut être intégré dans une plate-forme d'intégration continue tel que Jenkins. *Le sujet d'intégration continue n'est pas abordé dans ce document.*



Cette organisation de SVN comprend :

- un répertoire "plugins" pour les modules JCMS. Tous les développements JCMS doivent être effectués dans les modules. Pour plus de détails, consultez l'article dédié à ce sujet.
- un répertoire "conf" contenant toutes les configurations nécessaires destinées à chaque environnement (développement, recette, preprod, prod). La gestion des fichiers de configuration sera abordée dans la suite. Les fichiers de configuration de JCMS sont typiquement ceux-ci :
  - custom.prop
  - webapp.prop
  - web.xml
- un répertoire "data" qui contient le fichier "store.xml". La gestion du store sera détaillée dans la suite
- un répertoire "releaseJCMS" qui contient les releases JCMS : jcms-7.0.war, jcms-7.1.1.war, etc... Ce sont des nouvelles versions diffusées par Jalios. Dans la terminologie de JCMS, ces webapps sont souvent qualifiées sous les termes : "webapp

de base" ou "webapp vierge". En effet, elles ne contiennent aucun développement spécifique. Le stockage de ces versions sous forme compressée permet d'économiser l'espace et le temps de transfert de SVN vers le poste de développeur par rapport à un stockage décompressé.
En intégrant les modules développés spécifiquement dans ces webapps de base, le processus de construction pourra être validé en recette, en pré-production puis en production.

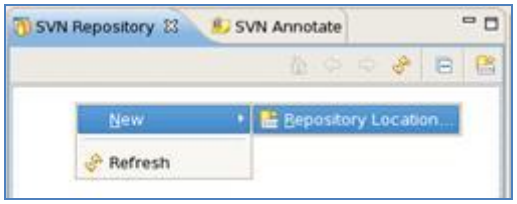
3.1.3. Eclipse et SVN

Jalios recommande l'utilisation de Eclipse pour développer avec JCMS. Afin de faciliter les tâches d'installation d'un environnement Eclipse, Jalios met à disposition un DevPack qui est un environnement prêt à l'emploi. Cet environnement regroupe les plugins Eclipse qui sont utiles comme :

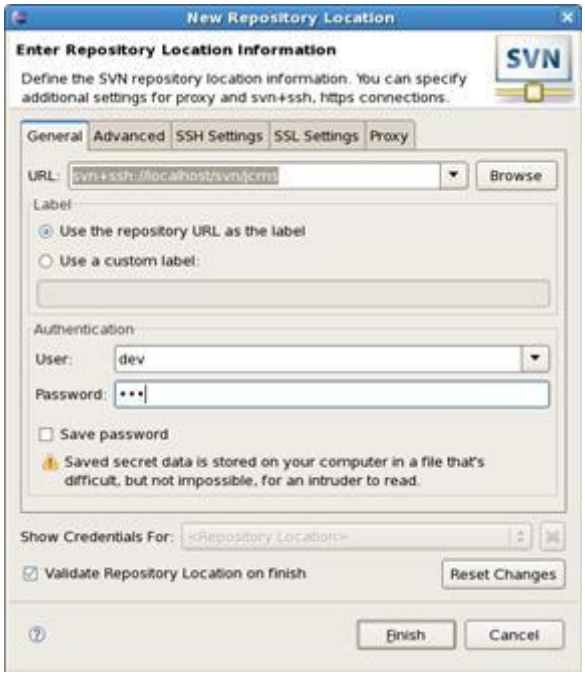
- Subversive qui ajoute le support de SVN dans Eclipse
- WTP (Web Tools Platform) qui facilite le développement des applications web et notamment l'amélioration de l'auto-complétion de code
- Jalios Plugin Tools (fourni par Jalios) qui ajoute le support de développement orienté module. L'utilisation de ce module sera détaillé dans la suite

Ajoutez un nouveau repository SVN dans Eclipse :

- Ouvrez la perspective "SVN Repository Exploring"
- Dans la vue "SVN Repository", cliquez avec le bouton droit dans la vue et choisissez "New > Repository Location…"



- Saisissez l'adresse du serveur SVN (e.g : svn+ssh://localhost/svn/jcms)
  Entrez l'utilisateur et le mot de passe pour accéder au serveur SVN (e.g. : dev/dev)

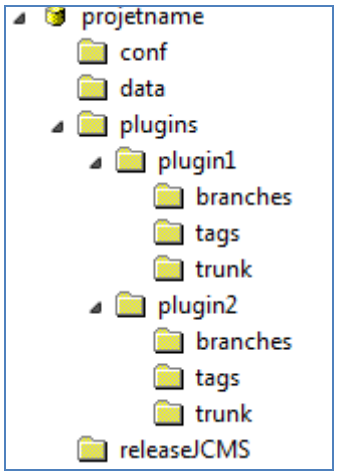


3.1.4. Check-in initial

Objectif : Récupération des modules, des données et de la configuration puis check-in initial de tous ces éléments sur le serveur SVN. Cette tâche est assurée par le mainteneur / responsable de développement.
En tant que responsable de développement, deux cas peuvent vous se présenter :

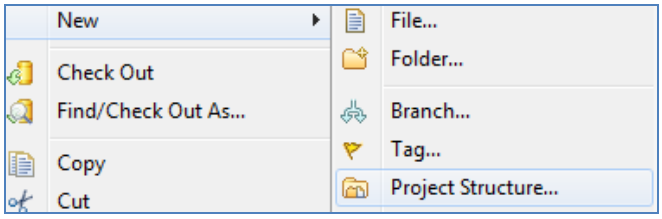
1. Il s'agit d'un nouveau projet. Dans ce cas, il faut connaître :
    - la version JCMS sur laquelle vous allez développer
    - les modules standards JCMS à utiliser / installer
    - les modules spécifiques à développer
2. Il s'agit de la maintenance d'un projet existant pour lequel il n'y a pas eu de gestionnaire de versions. Le seul élément disponible est la webapp JCMS de production. Dans ce cas, il faut déterminer :
    - la version JCMS sur laquelle le développement a été effectué. Pour ce faire, récupérez la webapp, la lancez et consultez la version JCMS indiquée dans l'état du site.
    - les modules standards JCMS à utilisées / installées : lancez la webapp et consulter la gestion des modules.
    - les modules spécifiques développés : c'est l'opération la plus délicate et coûteuse. Il est recommandé de consulter Jalios pour avoir un accompagnement sur le sujet.

Dans la suite, nous considérons que le 2ème cas peut se ramener au 1er cas et que vous avez tous les éléments nécessaires pour mettre sur SVN. Le but est d'arriver à l'arborescence ci-dessous :

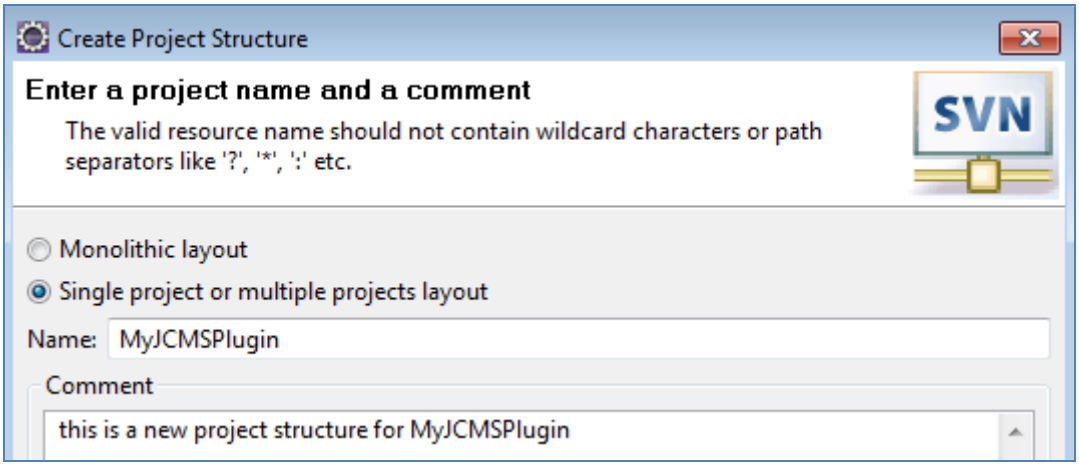


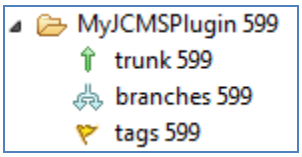
3.1.4.1. Check-in initial des modules

- Sur SVN, créez d'abord un répertoire "plugins". Cliquez droit sur le nom du projet > New Folder
- Créez ensuite les structures de projet. Cliquez droit sur "plugins" > New Project Structure

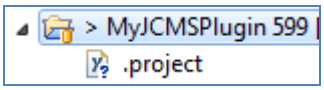


- Donnez un nom au nouveau module JCMS. Dans ce cas, c'est "MyJCMSPlugin".

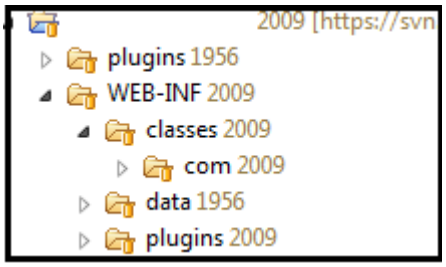

Create Project Structure
Enter a project name and a comment
The valid resource name should not contain wildcard characters or path separators like '?', '*', ':' etc.
SVN
Monolithic layout
Single project or multiple projects layout
Name: MyJCMSPlugin
Comment
this is a new project structure for MyJCMSPlugin



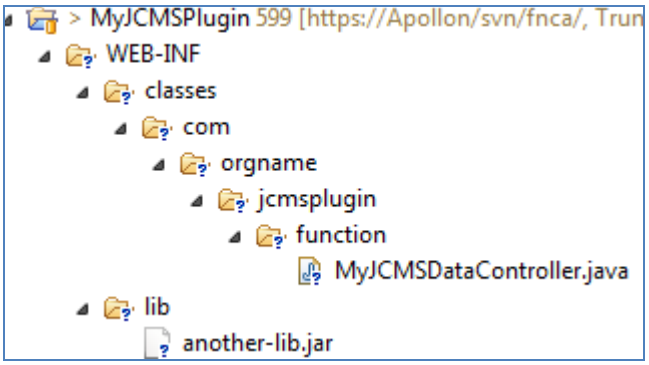
- Cliquez droit sur "trunk" puis sélectionnez "Check Out". Cette opération crée un projet Eclipse en utilisant le nom "MyJCMSPlugin". Pour vérifier que le checkout s'est bien déroulé, passez dans la perspective "Java". Un projet vide avec un fichier de configuration ".project" est présent.



- Ensuite, il faut importer une structure de module JCMS dans ce projet qui sera un projet module JCMS. Pour ce faire :
  - Soit vous possédez déjà un module JCMS sous forme de .zip. Dans ce cas, il suffit de le décompresser à la racine de "MyJCMSPlugin". L'arborescence ressemblera à la structure suivante :



  - Soit vous démarrez le développement, il faut créer ce module "MyJCMSPlugin". Dans ce cas, il faut aller dans l'interface web de JCMS puis créer/empaqueter un module "MyJCMSPlugin". JCMS va produire un module sous forme de .zip. Décompressez ce zip à la racine de "MyJCMSPlugin".
  - Au fur et à mesure du développement, vous aurez besoin d'implémenter des classes Java ou utiliser des librairies tiers dans votre module. Lors de l'empaquetage d'un module, JCMS produire un .zip contenant des .jars. S'il s'agit des .jars tiers dont vous ne possédez pas de sources, il suffit de les versionner tel quel dans SVN. Cependant, si c'était votre .jar, il faut versionner les sources. Votre .jar se trouvera sous le nom "MyJCMSPlugin.jar". Pour mettre les sources dans la structure du projet :
    - Créez un répertoire "WEB-INF/classes" en dessous du projet "MyJCMSPlugin"
    - Extrayez les sources .java dans "MyJCMSPlugin.jar" et les mettez en dessous de "WEB-INF/classes".
    - Attention, versionnez seulement les .java**. Il est inutile de le faire avec les .class**. On retrouve généralement l'arborescence suivante :



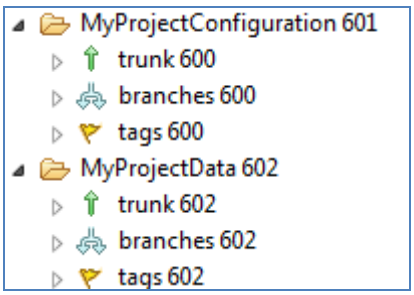
- Les fichiers peuvent être committés sur SVN pour la 1ère fois. Cliquez droit sur "*MyJCMSPlugin* > *Team* > *Commit*" puis valider l'opération.

Note : Dans la terminologie d'Eclipse, il s'agit d'un projet de type "General > Project", c'est un projet standard sans aucune nature supplémentaire. Eviter de créer des projets de type "Java Project" par exemple.

3.1.4.2. Check-in initial des configurations et des données

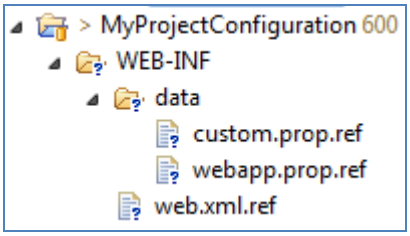
Objectif : Créer un référentiel pour les données. C'est le même principe de création des projets pour les modules JCMS :

- Créez un projet SVN "MyProjectConfiguration" pour le projet de configuration
- Faites de même pour le projet SVN "MyProjectData".
- Lancez le check-out sur le "trunk" de ces projets



Projet de configuration

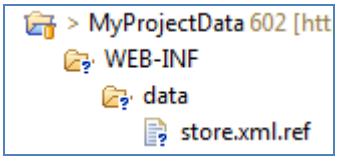
- Créez l'arborescence de répertoire "WEB-INF" puis "data" contenant deux fichiers "custom.prop.ref" et "webapp.prop.ref" avec une extension ".ref" (référentiel).
- En dessous de "WEB-INF", si votre projet JCMS nécessite de modifier le fichier "web.xml" (configuration NTLM, etc...), il faut le nommer en "web.xml.ref" également.
- Commitez ensuite ces fichiers.



Projet de données
Il y a trois zones de stockages les données dans JCMS :

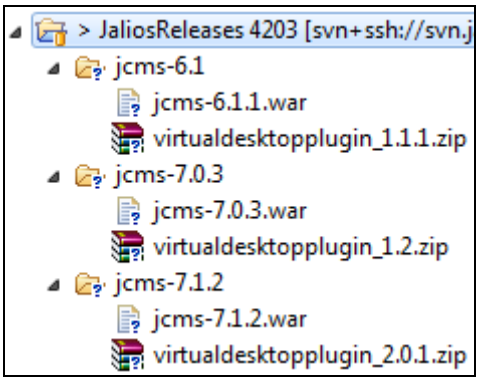
- stockage dans le store. Ce fichier se trouve dans "WEB-INF/data" sous le nom "store.xml". **Seul le store doit être versionné** dans SVN car il contient la structure des données.
- stockage dans la base de données JcmsDB. Ce sont des données utilisateurs qui ne doivent pas être reprises.
- stockage dans le répertoire "upload". Ce sont des données utilisateurs qui ne doivent pas être reprises

Pour le projet de données à versionner dans SVN, la structure est la suivante :



3.1.4.3. Check-in initial des wars JCMS et modules associées

Il s'agit du répertoire de stockage des releases Jalios permettant de gérer les constructions (ou les "builds") de livraison. La structure comporte un répertoire par version. Ce répertoire contient le war de JCMS et les modules JCMS associées à cette version.



Cette structure permet à un nouveau développeur de retrouver facilement les éléments nécessaires pour configurer son environnement.

3.1.5. Configuration initiale de l'environnement de développement

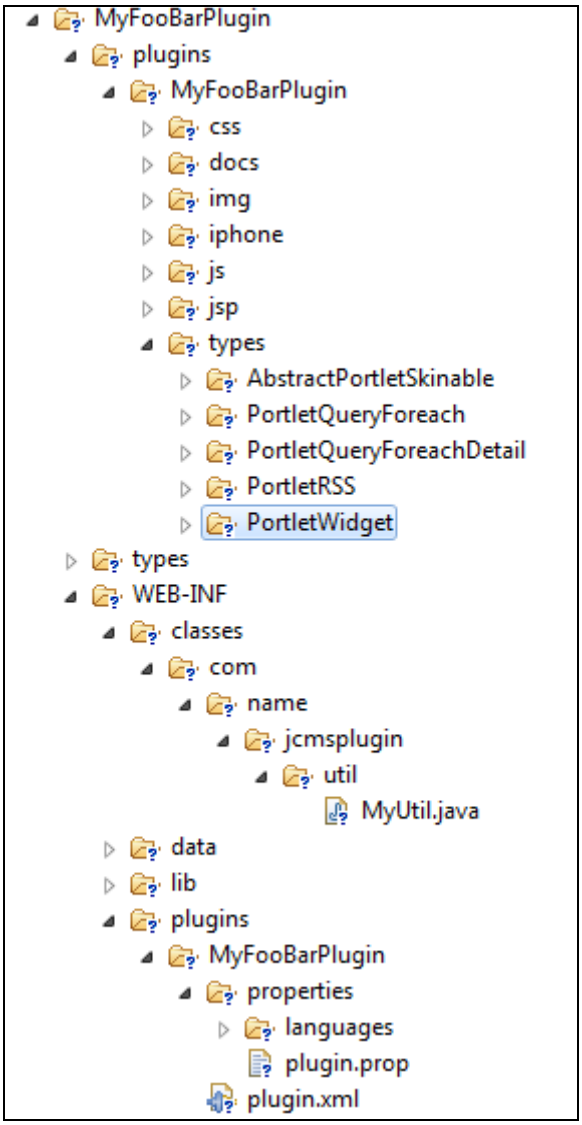
Objectif : Configuration l'environnement de développement JCMS sous Eclipse et récupération des composants nécessaires depuis le SVN. Cette tâche est réalisée par le développeur.

Contexte : Généralement, le mainteneur (ou le responsable de développement) va mettre à disposition une procédure permettant à un nouveau développeur de configurer son environnement lors de son arrivée sur le projet. Cette section va aborder cette procédure qui comporte trois étapes :

1. Checkout initial des projets de modules spécifiques JCMS
2. Checkout initial des configurations et des données
3. Création d'un projet web dynamique JCMS

3.1.5.1. Checkout initial des projets modules spécifiques JCMS

- Allez au repository SVN du projet communiqué par le responsable de développement. Sous Eclipse, ouvrez la perspective *"SVN Repository Exploring"* > *New* > *Repository Location*
- Choisissez le projet SVN du module à développer, par exemple "MyFooBarPlugin" dans notre cas. Cherchez ce projet sur SVN puis son "trunk".
- Lancez le check-out sur le projet. La structure, qui est un layout standard d'un module JCMS, ressemblera donc à celle-ci après le check-out :



Note : la configuration du projet est automatiquement récupérée lors du checkout. En effet le fichier ".project" d'Eclipse a été inclus volontairement dans le SVN lors de l'import initial par le mainteneur. Ainsi Eclipse ne propose pas de choisir le type de projet.

3.1.5.2. Checkout initial des configurations et des données

C'est le même principe que le checkout initial des projets modules :

1. Allez au repository SVN communiqué par le responsable de développement. Sous Eclipse, ouvrez la perspective "*SVN Repository Exploring*" > *New* > *Repository Location*
2. Lancez le check-out des projets de configurations et de données. Renseignez auprès du responsable de développement pour l'endroit de ces projets.

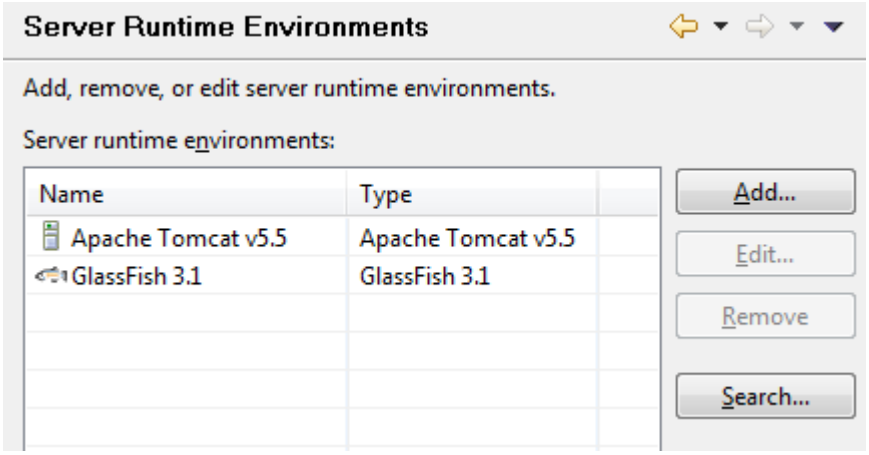
3.1.5.3. Création d'un projet web dynamique JCMS

Objectif : Configuration d'un projet web dynamique permettant de lancer une webapp JCMS.

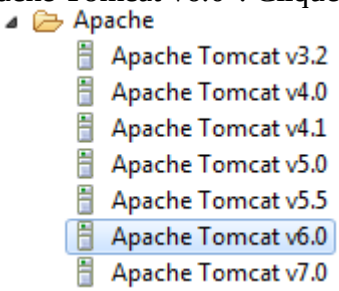
Contexte : La plate forme WTP (Web Tools Platform) de Eclipse propose plusieurs supports pour le développement d'une webapp JEE. Nous allons en profiter pour développer JCMS. WTP est inclut dans le package "Eclipse IDE for Java EE Developers" qui est celui embarqué dans le DevPack fourni par Jalios. Si vous n'utilisez pas le DevPack, il faut télécharger ce package Eclipse contenant WTP.

1. Configuration de l'environnement du serveur d'application à utiliser

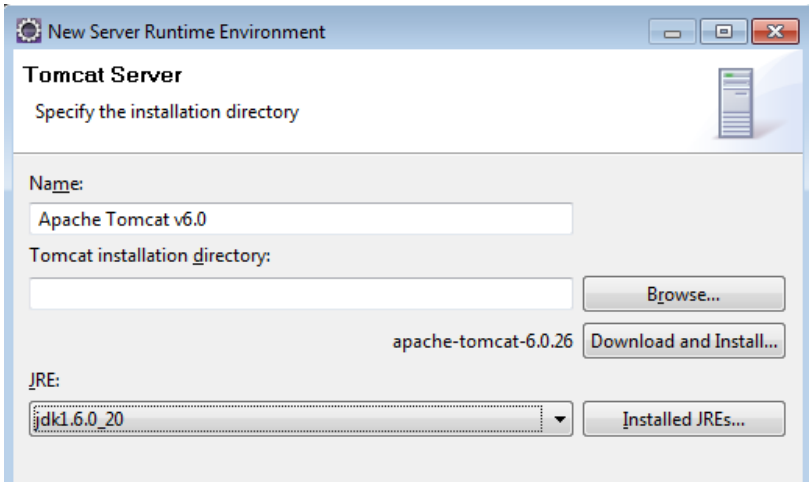
- Allez dans *Windows* > *Preferences* > *Server* > *Runtime Environnement*



- Si le serveur d'application cible n'existe pas encore, cliquez sur "Add". Ajoutez un serveur d'application "Apache Tomcat v6.0". Cliquez sur "Next"

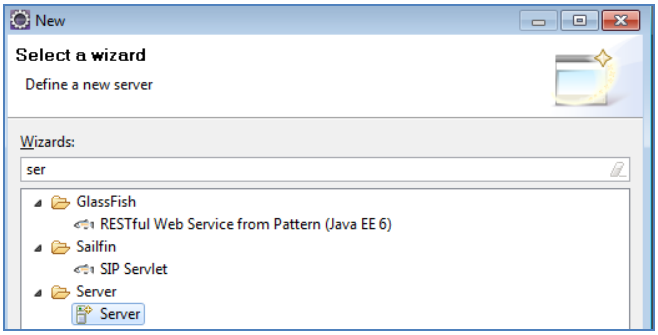


- Choisissez le répertoire d'installation de Apache Tomcat, par exemple v6.0 dans cet exemple. Il est recommandé d'utiliser la version préconisée par Jalios qui est précisée dans le manuel d'installation et d'exploitation.
- Associez également la version JDK. Nous recommandons de spécifier toujours la version exacte et d'éviter l'option "Workbench default JRE".

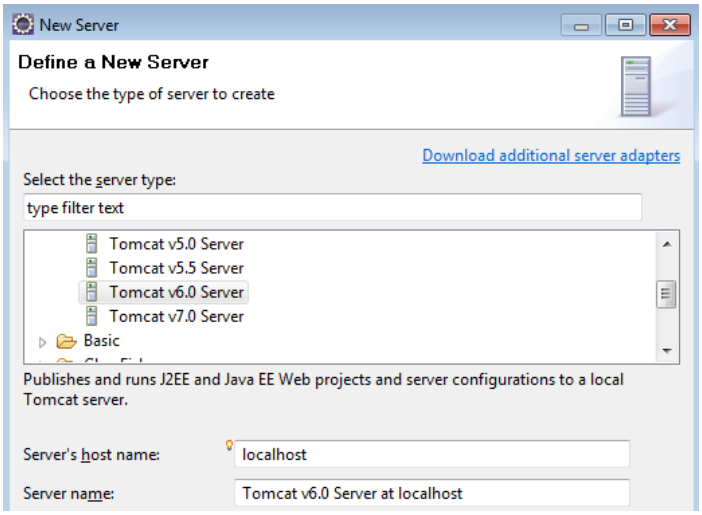


2. Création d'un lanceur de serveur d'application

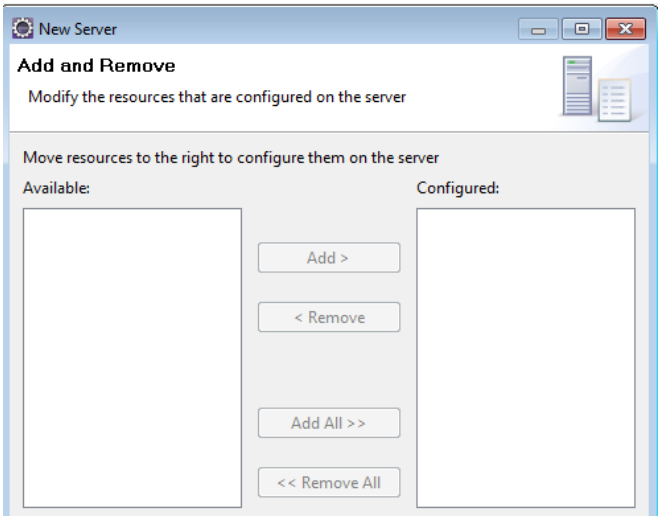
- Dans la perspective Java, *cliquez droit > New > Server > Faire "Next"*



- Sélectionnez le type du serveur. Dans notre exemple, choisissez *Tomcat v6.0 Server*. Cliquez sur "Next" pour continuer.

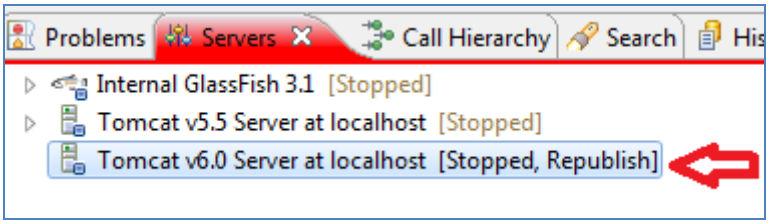

New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Select the server type:
type filter text
Tomcat v5.0 Server
Tomcat v5.5 Server
Tomcat v6.0 Server
Tomcat v7.0 Server
Basic
Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.
Server's host name:
localhost
Server name:
Tomcat v6.0 Server at localhost

- Pour terminer la configuration, Eclipse propose de configurer les projets web qu'il reconnaît. Laissez vide pour l'instant et cliquez sur "Finish".

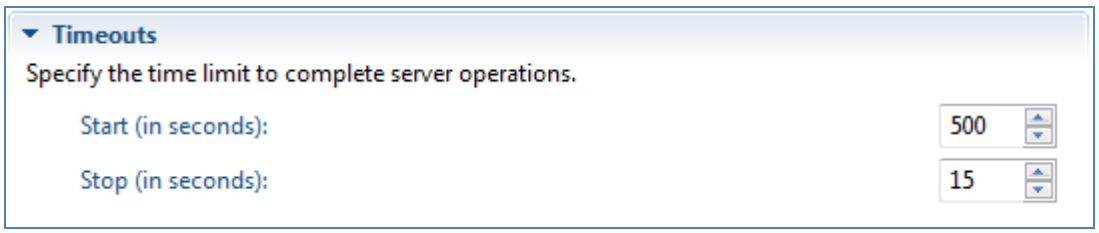


3. Configuration d'un lanceur de serveur d'application

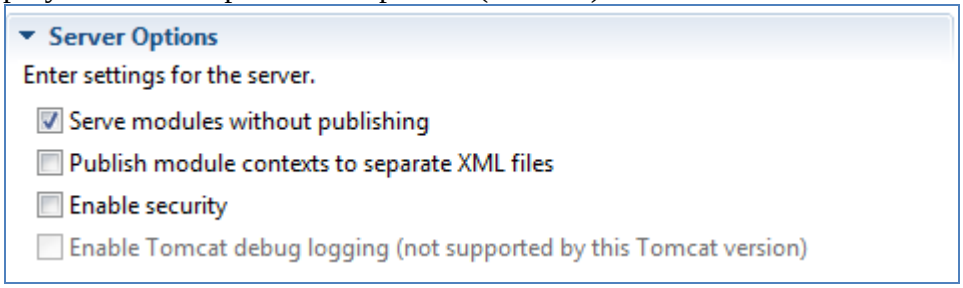
- Ouvrez la vue "Servers". Faites : *Window > Show View > Other > Chercher "Servers"*



- Double-cliquez sur "*Tomcat v6.0 Server at localhost*". Un récapitulatif des configurations va s'afficher.
- Dans la zone "Timeouts" se trouve le paramétrage qui déclenche l'arrêt du démarrage. Configurez le champ "Start" à 500s. Si le temps de démarrage dépasse 500s, le serveur va s'arrêter automatiquement.

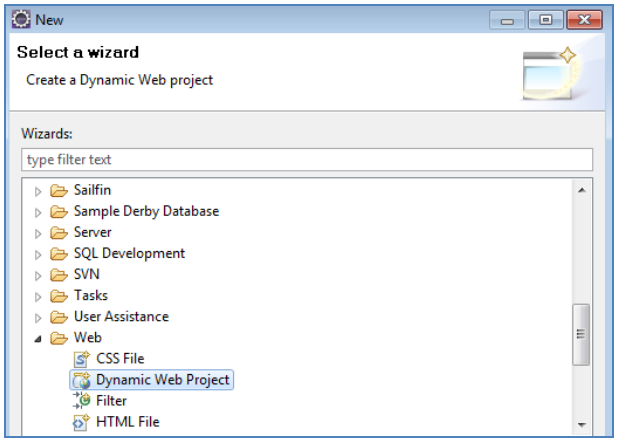


- La zone "*Servers Options*". Cochez la case "*Serve modules without publishing*". Cela permet à WTP de lire directement le répertoire "WebContent" du projet "Dynamic Web Project" qui sera créé par la suite. L'avantage est d'éviter de tout compresser et de déployer dans un répertoire temporaire (très lent).

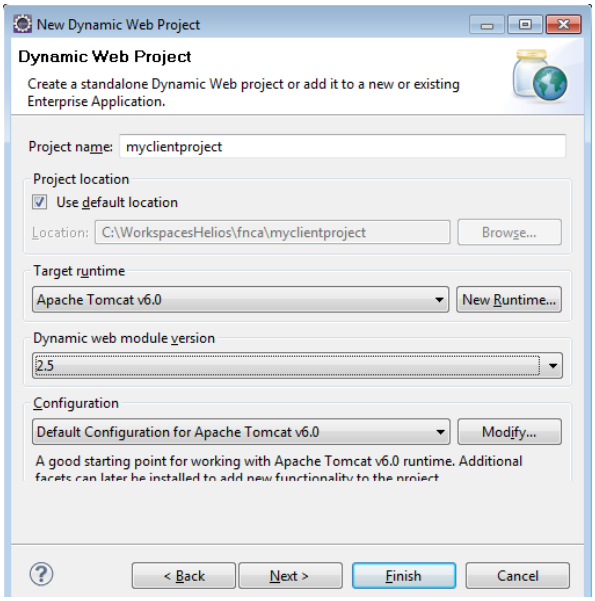

Server Options
Enter settings for the server.
Serve modules without publishing
Publish module contexts to separate XML files
Enable security
Enable Tomcat debug logging (not supported by this Tomcat version)

4. Création d'un projet "Dynamic Web Project"

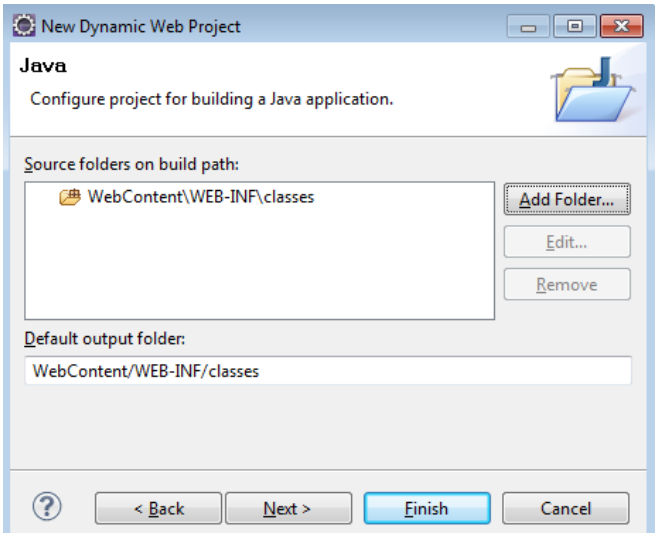
- Dans la perspective Java, *cliquez droit > New > Other > Cherchez "Dynamic Web Project" > "Next"*



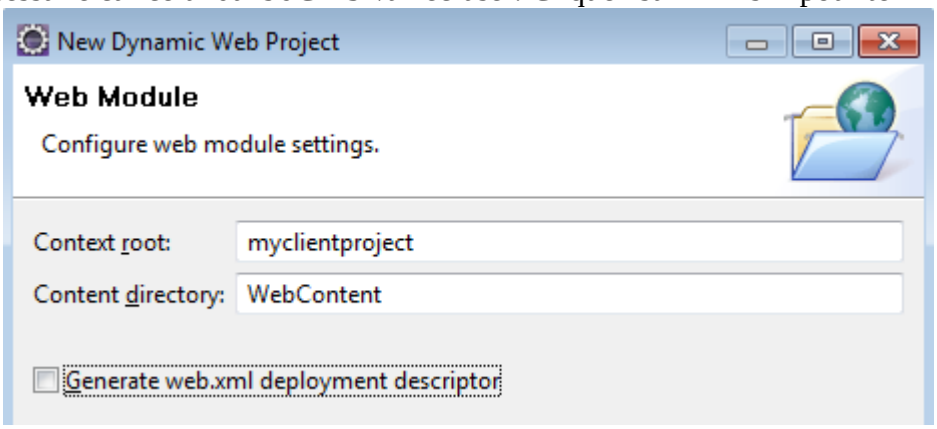
- Donnez un nom au projet. Par exemple "myclientproject". Cliquez sur "Next" pour continuer.



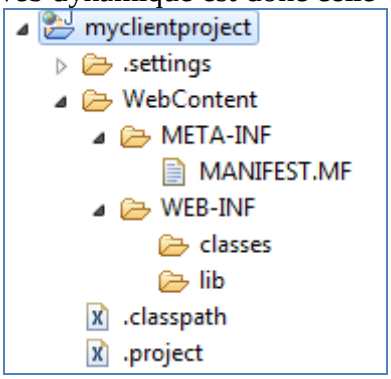
- Dans le champ "*Default output folder*", mettez "*WebContent/WEB-INF/classes*".
- Editez le champ "*Source folders on build path*" et mettez la même valeur. Cliquez sur "Next" pour continuer.



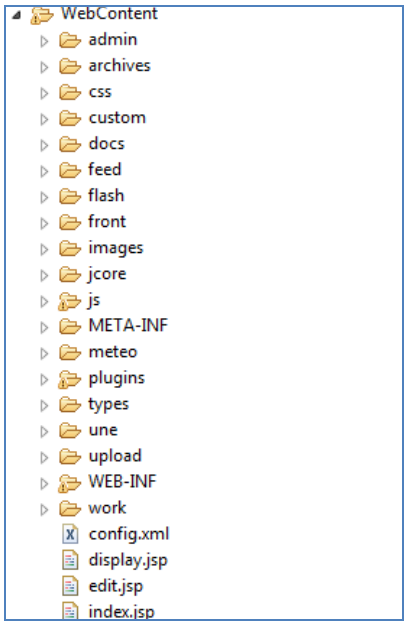
- Décochez la case "*Generate web.xml deployment descriptor*". Ce fichier n'est pas nécessaire car celui dans JCMS va l'écraser. Cliquez sur "Finish" pour terminer.



- La structure d'un projet web dynamique est donc celle-ci :



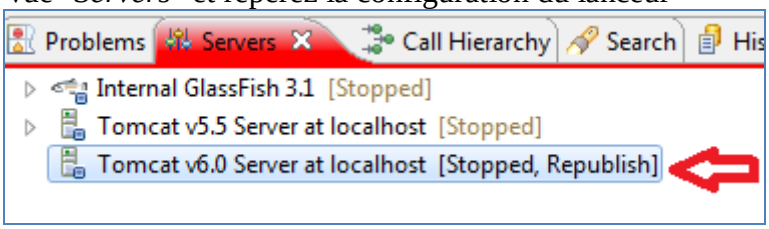
- Maintenant, cliquez droit sur le répertoire "WebContent" puis faites "*Import*" > *Archive File*. Choisissez le .war de JCMS qui est stocké dans le projet JaliosReleases précédemment évoqué dans la section 3.1.4.3
- La structure de la webapp JCMS en dessous du répertoire "WebContent" après l'import est la suivante :



5. Lancement d'un projet "Dynamic Web Project"

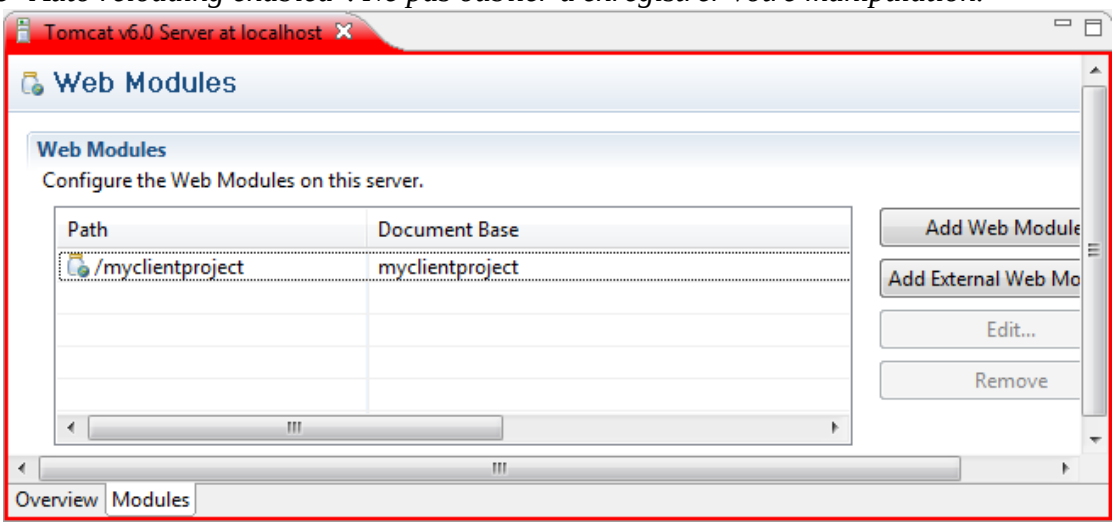
Il existe plusieurs manières pour lancer le projet "myclientproject". Une des façons est la suivante :

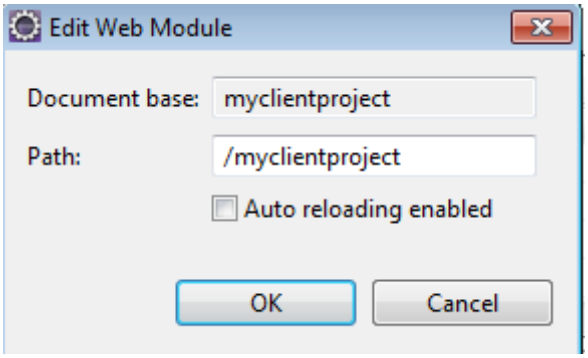
- Ouvrez la vue "*Servers*" et repérez la configuration du lanceur



- Cliquez droit sur *"Tomcat v6.0..." > Add and Remove*
- Choisir le projet "myclientproject"
- Cliquer droit sur *"Tomcat v6.0..." > "Start" ou "Debug"*

Astuce : Pour pouvor développer "à chaud", i.e sans relancer le serveur à chaque fois, vous pouvez éditer la configuration du lanceur. Dans l'onglet "Modules" > Edit > Décocher la case "Auto reloading enabled". Ne pas oublier d'enregistrer votre manipulation.





Résumé :
Dans cette section, nous avons pu voir comment créer un projet web dynamique supporté par WTP afin de lancer une webapp JCMS de base. Pour les autres modules standards de JCMS, procédez à l'installation standard en utilisant l'interface web de JCMS. Ensuite, rafraîchissez le projet web dynamique puis redémarrez.

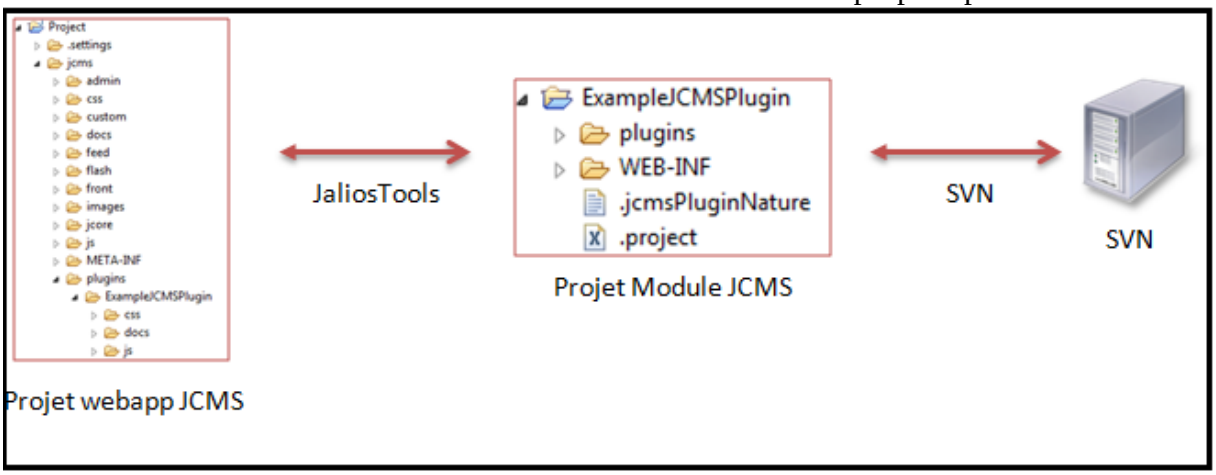
3.2. Développement

3.2.1. Synchronisation entre le projet module JCMS et la webapp JCMS

Contexte : A la fin de l'étape de check-out initial, nous avons :

- un ensemble des projets modules JCMS. Ces projets modules représentent un module développé spécifiquement.
- un ensemble des configurations et des données
- une webapp JCMS configurée sous la forme d'un projet "Dynamic Web"

Objectif : Versionner le développement module par module et développer de manière flexible.
Pour ce faire, le principe sera de synchroniser le projet module JCMS et le projet webapp JCMS d'où le besoin d'avoir un outil. Cette section décrit cet outil proposé par Jalios.

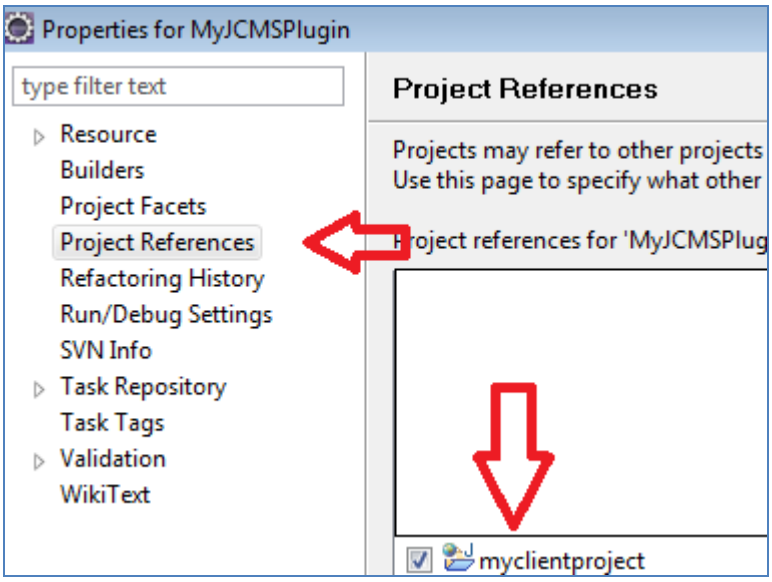


3.2.2. Jalios Plugin Tools

Jalios Plugin Tools est un module Eclipse. Il permet de synchroniser un projet module JCMS et un projet webapp JCMS. Il est proposé par défaut dans les DevPack JEE JCMS.

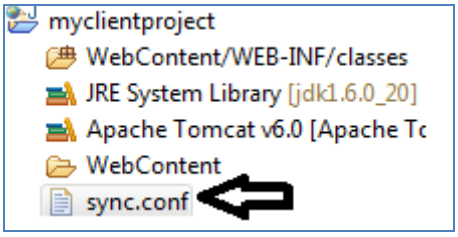
3.2.2.1. Configuration du projet module JCMS

- Pour que Jalios Plugin Tools puisse reconnaître un projet module JCMS, créez un fichier vide ".jcmsPluginNature" qui se situe au même niveau que celui ".project" standard d'Eclipse. Pour voir ces fichiers, passez dans la vue "Navigator" (*Windows* > *Show View* > *Navigator*).
- Ensuite, liez le projet module JCMS à celui de la webapp JCMS en utilisant la notion de référence standard dans Eclipse. Pour ce faire, cliquez droit sur le projet module *JCMS* > *Properties* > *Project References*. Cochez le projet webapp JCMS



3.2.2.2. Configuration du projet webapp JCMS

- Il s'agit du projet dynamique web que nous avons créé à la section précédente. Dans ce projet, créez un fichier "sync.conf" **à la racine du projet.**

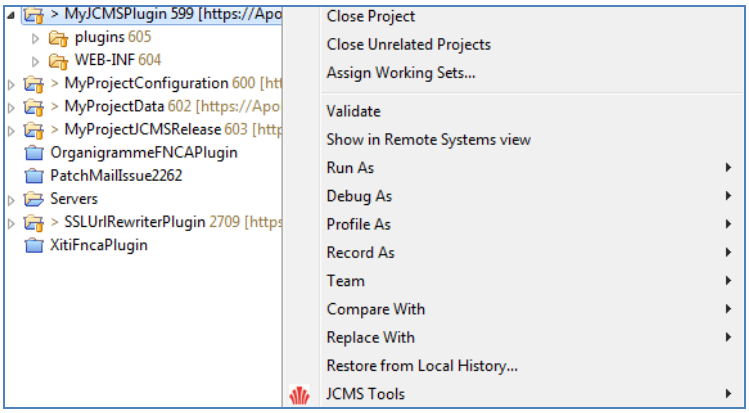


- Remplissez ce fichier avec le contenu suivant. Un exemple de ce fichier est fourni dans le DevPack. Les lignes précédées par "#" sont des commentaires. Ici, la propriété "root.dir" indique à Jalios Plugin Tools de synchroniser à partir du répertoire "WebContent" qui est considéré comme la racine.

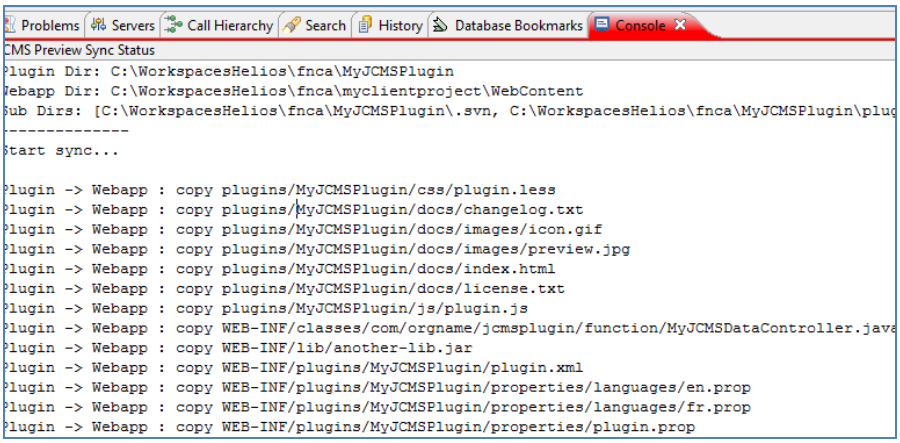
```
# The JCMS webapp root directory to sync
root.dir = WebContent
# Excluded directories from sync separated by a comma ","
# By default, .svn directory will be ignored from sync
# .settings is recommanded to be ignored too
excluded.dirs = .settings
# Excluded files from sync separated by a comma ","
# excluded.files = file1.jsp, file2.txt, file
```

3.2.2.3. Tester la synchronisation

- Cliquez avec le bouton droit sur le projet module JCMS pour afficher un menu contextuel



- Cliquez sur "*JCMS Tools* > *Sync Preview*" pour prévisualiser les actions que va effectuer la synchronisation.
- Cliquez sur "*JCMS Tools* > *Run Sync*" pour lancer la synchronisation.
- Dans le log du console d'Eclipse, les traces d'exécutions sont :
    - "Plugin => Webapp" signifie qu'il y a eu des copies à partir du projet module JCMS vers la webapp. Il s'agit dans ce cas probablement d'une mise à jour provenant de SVN par un autre développeur.
    - "Webapp => Plugin" signifie le sens inverse. Il s'agit donc d'une fin de développement et que le développeur décide de committer sur SVN


Problems
Servers
Call Hierarchy
Search
History
Database Bookmarks
Console
CMS Preview Sync Status
Plugin Dir: C:\WorkspacesHelios\fnca\MyJCMSPlugin
Webapp Dir: C:\WorkspacesHelios\fnca\myclientproject\WebContent
Sub Dirs: [C:\WorkspacesHelios\fnca\MyJCMSPlugin\.svn, C:\WorkspacesHelios\fnca\MyJCMSPlugin\plu
Start sync...
Plugin -> Webapp : copy plugins/MyJCMSPlugin/css/plugin.less
Plugin -> Webapp : copy plugins/MyJCMSPlugin/docs/changelog.txt
Plugin -> Webapp : copy plugins/MyJCMSPlugin/docs/images/icon.gif
Plugin -> Webapp : copy plugins/MyJCMSPlugin/docs/images/preview.jpg
Plugin -> Webapp : copy plugins/MyJCMSPlugin/docs/index.html
Plugin -> Webapp : copy plugins/MyJCMSPlugin/docs/license.txt
Plugin -> Webapp : copy plugins/MyJCMSPlugin/js/plugin.js
Plugin -> Webapp : copy WEB-INF/classes/com/orgname/jcmsplugin/function/MyJCMSDataController.java
Plugin -> Webapp : copy WEB-INF/lib/another-lib.jar
Plugin -> Webapp : copy WEB-INF/plugins/MyJCMSPlugin/plugin.xml
Plugin -> Webapp : copy WEB-INF/plugins/MyJCMSPlugin/properties/languages/en.prop
Plugin -> Webapp : copy WEB-INF/plugins/MyJCMSPlugin/properties/languages/fr.prop
Plugin -> Webapp : copy WEB-INF/plugins/MyJCMSPlugin/properties/plugin.prop

3.2.2.4. Limites

La synchronisation du module Eclipse "Jalios Plugin Tools" connaît les limites suivantes :

- suppression de fichier :
    - si un développeur s'aperçoit qu'un fichier n'est plus utile, il le supprime puis teste dans la webapp. Lors de la synchronisation, le fichier ne sera pas supprimé mais recopié depuis le projet module vers le projet webapp.

    - inversement, si un fichier est supprimé sur le serveur SVN. Lors de son update, le développeur va s'apercevoir qu'il y a eu une suppression. Dans ce cas, lors de la synchronisation, le fichier sera copié de la webapp vers le projet module.

    - actuellement, dans tous les cas, quand il s'agit d'une suppression, il faut procéder manuellement à la suppression du fichier **dans les deux projets**.

- détection des nouveaux fichiers :
    - seul le répertoire "*plugins/MyJCMSPlugin*" est scanné pour tester s'il y a des nouveaux fichiers créés.
    - pour tous les autres fichiers créés en dehors de ce répertoire (une classe java, un type, un workflow, etc...), il faudrait procéder à une copie manuelle vers le projet module.

3.2.3. Gestion du fichier custom.prop

Si le fichier custom.prop contient des modifications à reporter dans le svn (pour tous les développeurs), fusionnez manuellement le fichier custom.prop et le fichier custom.prop.ref. Sélectionnez au cas par cas les propriétés à récupérer.
Attention, le fichier custom.prop n'est pas déployé par le script de déploiement. Il est recommandé de mettre les propriétés spécifiques dans les modules (plugin.prop).

3.2.4. Synchronisation des données du store

Deux options se présentent après une synchronisation avec le repository :

Option 1 : Ecraser le store local en le remplaçant par le store de référence du SVN.

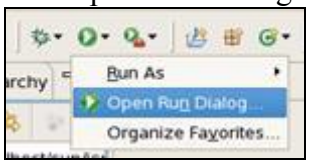
Optez pour cette option si vous ne souhaitez pas conserver les données (contenus, portlet, catégories, ...) que vous avez créé dans votre environnement de développement.

Option 2 : Fusionner le store local et le store de référence du SVN.

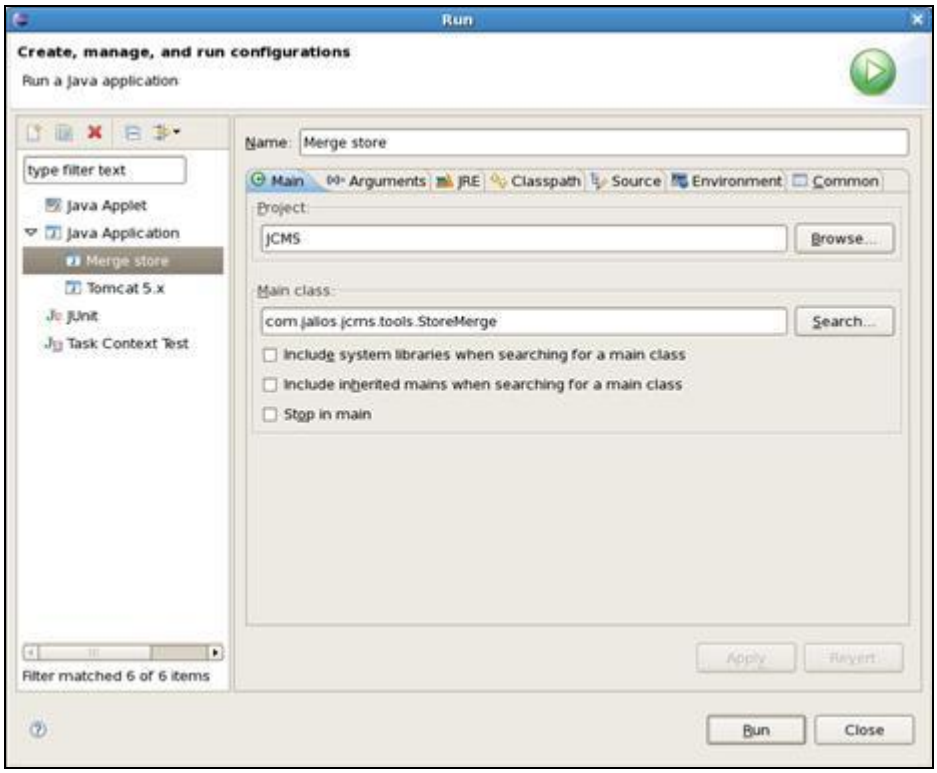
Optez pour cette option si vous avez créé des publications lors de vos développements et que vous souhaitez conserver ces publications tout en récupérant les éventuelles modifications disponibles dans le store de référence.
Les fusions des stores sont des options délicates. Jalios fournit l'outil Store Merge pour les réaliser.

Configuration du Store Merge

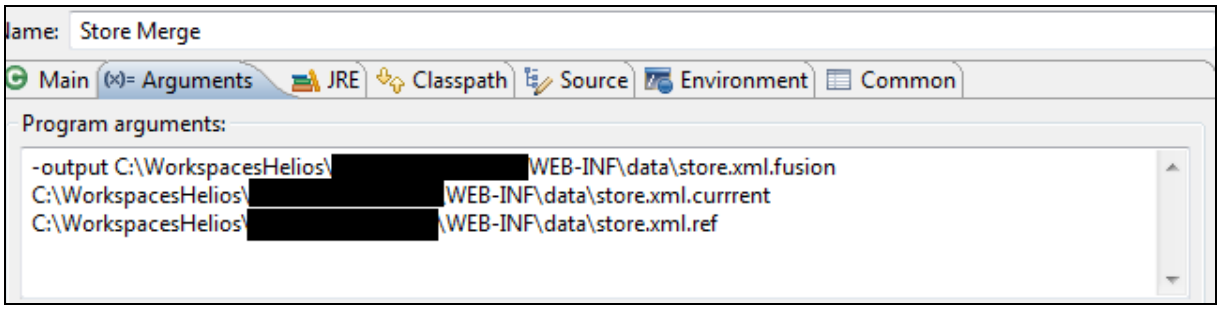
- Ouvrez la perspective Java.
- Dans le menu "Run" choisissez "Open Run Dialog…"



- Créez une nouvelle application Java.
- Dans l'onglet "Main" spécifiez :
    - Votre projet principal (e.g. JCMS) dans le champ "Project"
    - Saisissez com.jalios.jcms.tools.StoreMerge dans le champ "Main class"



- Allez ensuite dans l'onglet "Arguments" :
  - Spécifiez le chemin du store fusionné avec le paramètre -output
  - Spécifiez le chemin du store courant / local
  - Spécifiez le chemin du store référentiel



- Cliquez sur "Run" pour lancer la fusion.

Note : Les options avancées du Store Merge sont indiquées à l'annexe.

Si une fusion doit être effectuée, le Store Merge effectue les opérations suivantes :

1. Une copie de sauvegarde du store local est effectuée.
2. Le store de référence étant déjà dans le SVN, aucune sauvegarde n'est effectuée pour ce fichier.
3. Les opérations JStore des deux stores sont fusionnées dans un nouveau store via le paramètre -output

Voici les scénarios possibles et les actions que vous pouvez/devez effectuer par la suite.

a. Les deux stores sont identiques :
Sortie de la console :

```
INFO - current store              : WEB-INF\data\store.xml.current
INFO - ref store                  : WEB-INF\data\store.xml.ref
INFO - conflictResolution         :
INFO - ignoreConflicts            : false
INFO - strictUpdateConflicts      : false
INFO - useJStoreSyncOp            : false
INFO - useClassicIO               : false
INFO - gcs                        : (no gcs)
INFO - ouput                      : WEB-INF\data\store.xml.fusion
INFO - prepareConflictResolution  :
INFO - Start merging...
INFO - Compute store diff
INFO - Greatest Common Stamp: tl_7046
INFO - Load divergent suffixes...
INFO - Current store suffix: 0 operations on 0 data
INFO - Reference store suffix: 0 operations on 0 data
INFO - Check conflicts...
WARN - No diff. Stop merge.
INFO - The stores have not been changed (status: No Diff).
```

Les fichiers ne sont pas modifiés.

⇨ Vous pouvez commiter vos autres développements.

b. Le store local a été modifié, le store de référence n'a pas évolué :
Sortie de la console :

```
INFO - current store: webapp/WEB-INF/data/store.xml.current
INFO - ref store    : webapp/WEB-INF/data/store.xml.ref
INFO - Start merging...
INFO - Compute store diff
INFO - upgrade the store
INFO - compute the new suffix
INFO - Copy the newSuffix in refStore from deploy_5000
INFO - Merge done.
INFO - Override the ref store
INFO - Backup the current store: webapp/WEB-INF/data/store.xml-1234567890123.bak
INFO - Copy the new ref store as the current store
INFO - end merging.
```

Le store.xml.ref est remplacé par le store.xml local.

⇨ Si vous souhaitez envoyer vos données dans le SVN, vous pouvez commiter le nouveau store.xml.ref et vos autres développements.

c. Le store de référence a été modifié, le store local n'a pas évolué :
Sortie de la console :

```
INFO - current store: webapp/WEB-INF/data/store.xml
INFO - ref store    : webapp/WEB-INF/data/store.xml.ref
INFO - Start merging...
INFO - Compute store diff
```

```
INFO - upgrade the store
INFO - compute the new suffix
INFO - Copy the newSuffix in refStore from deploy_5000
INFO - Merge done.
INFO - Override the ref store
INFO - Backup the current store: webapp/WEB-INF/data/store.xml-1234567890123.bak
INFO - Copy the new ref store as the current store
INFO - end merging.
```

Le store.xml local est remplacé par le store.xml.ref.

⇨ Démarrez votre webapp avec le nouveau store de référence pour valider que vos développements fonctionnent avec ce nouveau store.
Vous pouvez commiter vos autres développements.

d. Les deux stores ont évolués et il n'y aucune opération conflictuelle

Sortie de la console :

```
INFO - current store: webapp/WEB-INF/data/store.xml
INFO - ref store    : webapp/WEB-INF/data/store.xml.ref
INFO - Start merging...
INFO - Compute store diff
INFO - upgrade the store
INFO - compute the new suffix
INFO - Copy the newSuffix in refStore from deploy_5000
INFO - Merge done.
INFO - Override the ref store
INFO - Backup the current store: webapp/WEB-INF/data/store.xml-1234567890123.bak
INFO - Copy the new ref store as the current store
INFO - end merging.
```

Les deux stores sont fusionnés dans un nouveau store spécifié via le paramètre -output

⇨ Démarrez votre webapp avec le nouveau store pour valider la fusion et pour valider que vos développements fonctionnent avec ce nouveau store.
Si vous souhaitez envoyer vos données dans le SVN, vous pouvez commiter le nouveau store.xml.ref et vos autres développements.

e. Les deux stores ont évolués et certaines opérations sont conflictuelles

Sortie de la console :

```
INFO - current store            : WEB-INF\data\store.xml.current
INFO - ref store                : WEB-INF\data\store.xml.ref
INFO - conflictResolution       :
INFO - ignoreConflicts          : false
INFO - strictUpdateConflicts    : false
INFO - useJStoreSyncOp          : false
INFO - useClassicIO             : false
INFO - gcs                      : (no gcs)
INFO - ouput                    : WEB-INF\data\store.xml.fusion
```

```
INFO - prepareConflictResolution   :
INFO - Start merging...
INFO - Compute store diff
INFO - Greatest Common Stamp: sa_5835
INFO - Load divergent suffixes...
INFO - Current store suffix: 460 operations on 460 data
INFO - Reference store suffix: 495 operations on 495 data
INFO - Check conflicts...
WARN - Conflict update/update on data 'sa_5173' (generated.PortletQueryForeach) for
attribute 'displayTitle'
WARN - Conflict update/update on data 'j_2' (member) for attribute 'opAuthor': "" / "j_2"
...
WARN - Conflict update/update on data 'tl_6547' (generated.CollaborativeSpace) for
attribute 'childrenBindings'
...
WARN - 13 conflicts detected on 6 data. Stop merge.
INFO - The stores have not been changed (status: Conflicts).
```

Un ou plusieurs conflits ont été détectés.

Procédure de résolution des conflits :

- Lancez à nouveau le Store Merge avec l'option -prepareConflictResolution en mentionnant le chemin du fichier de résolution à générer. Le fichier généré aura la forme suivante :

```
# Resolution options:
#    1: currStore
#    2: refStore

...
# Conflict update/update on data 'tl_6547' (generated.CollaborativeSpace) for attribute
'childrenBindings'
tl_6547: 1

# Conflict update/update on data 'j_2' (member) for attribute 'opAuthor': "" / "j_2"
j_2: 1
```

  Par défaut, c'est le store courant / local qui fait foi.
- Si c'est le store de référentiel qui fait foi, modifiez la valeur à 2.
- Une fois que les décisions de résolution de conflits sont prises, enlevez l'option -prepareConflictResolution et lancez le Store Merge avec l'option -conflictResolution en fournissant le fichier de résolution
- Démarrez la webapp avec le nouveau store.
- Lancez un contrôle d'intégrité pour valider la fusion. Pour les conflits de type update/update, vérifiez l'état des données concernées dans le nouveau store. Vous pouvez consulter l'historique pour avoir le détail des changements.

Si vous souhaitez envoyer vos données dans le SVN, vous pouvez commiter le nouveau store.xml.ref et vos autres développements.

3.3. Recette

Objectif : Déploiement des développements sur le serveur de recette et validation.

Lorsqu'un les développements sont terminés, en tant que mainteneur effectuez les opérations suivantes :

- Gelez les commits des développeurs (éventuellement via un lock sur le SVN)
- Synchronisez l'environnement avec le SVN.
- Ouvrez le « manuel d'installation et d'exploitation » et suivre le chapitre 8 sur "Mise à jour et de déploiement via le script de déploiement".
- Demandez aux équipes de recette de valider les développements.
- Attendez les retours de l'équipe de recette et effectuez les correctifs demandés.
- Si la recette est validée, effectuez un tag sur SVN.

3.4. Mise en production

Objectif : Déploiement de la webapp validée par la recette sur le serveur de production en utilisant le script de déploiement. Consultez le chapitre 8 sur "Mise à jour et de déploiement via le script de déploiement".

3.5. Annexe

3.5.1. FAQ

Pourquoi ne pas versionner toute la webapp JCMS ?
Versionner toute la webapp JCMS est coûteux pour un système de gestion de configuration. Cette approche n'est pas cohérente avec le mécanisme de développement orienté module de JCMS.

Pourquoi ne pas versionner partiellement la webapp JCMS, c'est à dire versionner le delta de changement ?
Cette approche n'est pas cohérente avec le mécanisme de développement orienté module de JCMS car elle ne permet pas de suivre l'évolution module par module.

Sur quel critère Jalios Plugin Tools se base-t-il pour effectuer la synchronisation ?
Actuellement, la synchronisation se base sur la date de la dernière modification qui détermine le sens de synchronisation (Projet Webapp <=> Projet module).

Comment éviter les doublons dans la recherche (Ctrl-Shift-R) ?
Cette approche de synchronisation entraîne des doublons dans la recherche des fichiers. Pour contourner, dans Eclipse, on peut définir la notion de "Working Set", on peut donc limiter la recherche dans l'arborescence webapp JCMS. Pour ce faire :

- Ctrl-Shift-R pour lancer la recherche
- Repérer la flèche "Menu" en haut à droite de la pop-up
- Sélectionner "Select Working Set..." puis suivre les indications d'Eclipse (par exemple un type working set Resource)

3.5.2. Usage du Store Merge

Usage :

```
usage: StoreMerge [options] store.xml store.xml.ref
```

Options :

-classicIO	use the classic IO (slower but consume less memory)
-conflictResolution <arg>	a file that contains a resolution for each conflicting ID
-gcs <arg>	provide the Greatest Common Stamp (exclusive with -useJStoreSyncOp option)
-ignoreConflicts	ignore the conflicts
-output <arg>	the output file. If this argument missing, store.xml.ref will be used as output.
-prepareConflictResolution	a file that will be filled with each conflicting ID

<arg>	
-strictUpdateConflicts	warn for each update/update conflict whatever the attribute values are.
-useJStoreSyncOp	use the <JSTORE:SYNC> mark to find out the store divergence (exclusive with -gcs option)

Exit code :

0: Merge done (or no merge to perform) 1: Bad parameter 2: Error during the merge 3: Conflicts have been detected